Calvin Guillot

Modular Sensor Environment

Audio Visual Industry Monitoring Applications

Helsinki Metropolia University of Applied Sciences

Electronics Engineering

Bachelor in Electronics

Thesis

August 2017

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Calvin Guillot Modular Sensor Environment 29 pages + 2 appendices 24 August 2017
Degree	Bachelor Degree of Engineering
Degree Programme	Degree Programme in Electronics
Specialisation option	-
Instructor(s)	Janne Mäntykoski, Senior Lecturer, Metropolia AMK Teemu Penttinen, Supervisor, Electro Waves Oy

This work was made for Electro Waves Oy. The company specializes in Audio-visual services and interactive systems.

The purpose of this work is to design and implement a modular sensor environment for the company, which will be used for developing automated systems.

This thesis begins with an introduction to sensor systems and their different topologies. It is followed by an introduction to the technologies used in this project.

The system is divided in three parts. The client, that consist of a device that has a HC SR-04 ranging sensor and an ESP8266 for the data processing; a server built with Node.js, Vue.js and MongoDB for gathering the data; and a dedicated application built on top of the server that presents the data to the end user.

The client connects to the server using UDP over Wi-Fi. This was necessary to allow modularity on the system. A 3D printed enclosure was designed and manufactured for the device, to create a production ready prototype that later will be commercialized. Finally, an operational application was built to showcase the behaviour of the system.

The proposed environment yielded satisfactory performance and design standards. This work will serve as a platform for future Electro Waves products.

Keywords	Modular, Sensor, HR-SC 04, ESP8266, JavaScript, HTML, 3D printing, AV applications, Automation, Electro Waves

i. List of figures

ii. List of tables

iii. Abbreviations

Contents

List of Figures

List of Tables

Abbreviations

ABS	Acrylonitrile Butadiene Styrene
AM	Additive Manufacturing
AP	Access Point
AV	Audio Visual
CAD	Computer Aided Design
CSS	Cascade Style Sheets
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name Servers
DOM	Document Object Model
EEPROM	Electrically Erasable Programmable Read-Only Memory
EW	Electro Waves Oy
FDM	Fused deposition modelling
GND	Ground
HTML	Hypertext Mark-up Language
I/O	Input Output
IC	Integrated Circuit
IDE	Integrated Development Environment
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
MC	Microcontroller
MSE	Modular Sensor Environment
NoSQL	Non-relational SQL
OSI	Open Systems Interconnection
PLA	Polylactic acid

RSS	Rich Site Summary
RSSI	Received Signal Strength Indicator
RST	Reset
SQL	Structured Query Language
SSID	Service Set Identifier
SVG	Scalable Vector Graphics
TCP	Transmission Control Protocol
TRIG	Trigger
UDP	User Datagram Protocol
UI	User Interface
USB	Universal Serial Bus
VCC	Voltage positive power supply
VPN	Virtual private network
WSN	Wireless Sensor Network
XML	Extensible Mark-up Language

1 Introduction

In the last few decades, we have seen the rapid evolution of devices that can capture data in a cheap and reliable way. These devices can vary greatly in size and form but all of them perform a basic function. They measure a physical quantity and transform it into a quantifiable measurement.

With the dawn of the internet of things, most of these sensors will be connected between each other into bigger and bigger networks which will allow to gather substantial information about people, weather, traffic, and so on. However, as the sensor networks grow, they tend to become over complicated due to the lack of standardization, expensive systems, and inefficient deployment.

The goal of this thesis was to develop a modular sensor environment (MSE) that was easy to manufacture, install, and customize for Electro Waves Oy (EW). This company specializes in Audio-Visual (AV) technologies, digital signage, lighting, and interactive control systems. As a part of their product catalogue, the company develops control systems for human interaction and AV remote control. They required a MSE which will be use in their digital signage and AV solutions, and as proprietary applications (See chapter 4), and eventually, home automation and other consumer services.

This work introduces a basic description of different MSE and sensor network topologies, different approaches of data acquisition, the ESP8266 board and the different technologies used for processing the data. Furthermore, 3D printing technologies will be discussed in brief including their advantages and limitation for manufacturing this kind of products.

2 Sensor Systems

There are many ways to acquire data from physical measurements. Current technology allows us to have several approaches to specific applications. One notorious modern approach is to assemble multiple sensor in a network to increase the reach of the system.

Sensors can gather data from almost any variable, from temperature, pressure and humidity, to electromagnetic interference sound waves. By combining difference sensors in a system, it is possible to have a virtual representation of the environment, and monitor changes within a chosen area.

2.1 Wireless Sensor Networks

A wireless sensor network (WSN) is an autonomous sensor system that measures any given variable in a specific environment. Modern systems are bi-directional and can have different topology arrangements. The main advantages of these systems are: the resilience of the system (the ability to work if 1 or more nodes fails); the modularity of the nodes, i.e. the nodes are all the same and work in the same way regarding of their location, unless the application requires different types of sensors per node; their scalability; and finally, their easy deployment.

A major concern in a WSN is the ability to produce nodes in a way that there are cost effective and can be easily replaced. This work proposes a cheap and straight-forward solution to this problem.

2.2 Network Topologies

A network topology describes the arrangement of the different elements within a communication network. This allows the system to cover a greater spatial area than a single node would. There are two categories of network topologies: physical and logical.

Physical topologies refer to the spatial arrangement of the nodes within the network and how the connect with each other, i.e. cabling and location. Logical topologies refer to the way this node communicate with each other.

A network can have different physical and logical topologies, or they can have the same topology depending of the application. Figure 1 illustrates the different topologies.

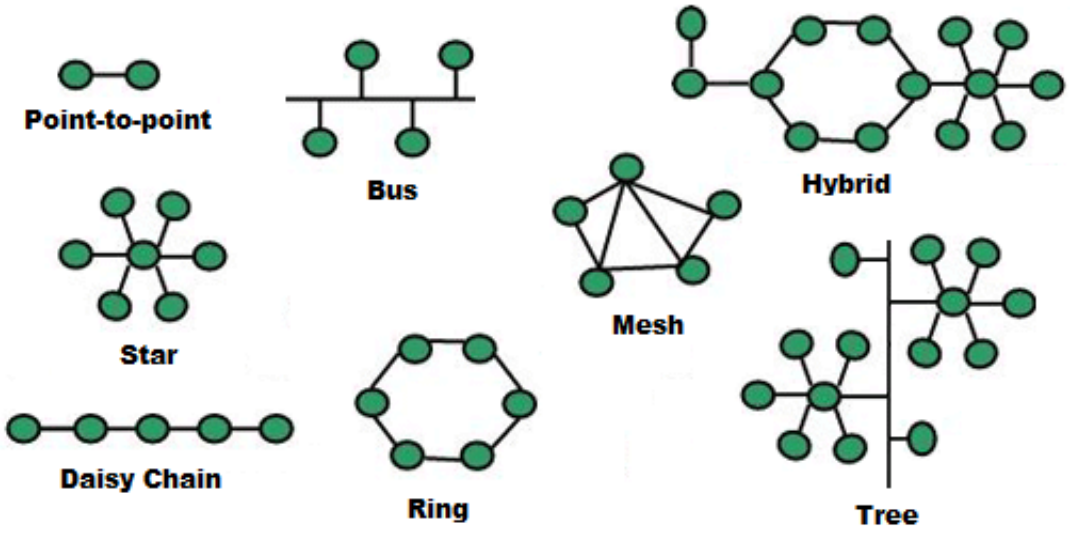

Figure 1 Network topology different configurations [1]

The network topologies can be classified in 8 basic categories [1]. Table 1 presents their characteristics, advantages and limitations.

Table 1 Network topologies characteristics

TYPE	ADVANTAGES	LIMITATIONS
Bus	Inexpensive	Single point of failure
Daisy-Chain	Linear communication	Expensive
Hybrid	Practical	Redundancy
Mesh	Full connectivity	Impractical at large scale
Point-to-point	Dedicated link	Only two nodes
Ring	No hierarchy	Bandwidth could be bottle-necked by the weakest link
Star	Easy to implement, scala-bility	Single point of failure
Tree	Modular	Multiple points of failure

This work uses the star topology for the proposed system. This arrangement favours centralization and control over dependency of the main gateway, given that if this fails or

disconnects from the network, all the nodes will be isolated and will not be able to communicate with each other or to the outside world. However, this arrangement assures that all the nodes are connected and transmitting.

2.3 Hardware

Modern integrated circuits (IC) are a low cost, easy to implement solution, that are used practically everywhere. The Internet of Things (IoT) is one of the fastest growing industry in recent years. It involves every kind of application which requires internet connection. From home automation, to weather monitoring, people tracking, and industrial applications such as assembly line monitoring and more. This work will focus on the development of a sensor system using small yet fast and reliable IC board and an ultrasonic ranging sensor.

2.3.1 ESP8266

The ESP8266 board is a low cost, IoT ready, Wi-Fi enabled microcontroller (MC). This board can be programmed using the Arduino IDE (see 2.4.1 Arduino) over a serial interface [2]. Originally develop with Expressif Systems [3], this board has become one of the most popular and useful MCs in the market. It can be programmed to control and trigger, relays, switches, sensors and other devices. The main feature of the board is its small size and Wi-Fi connectivity, which make it a perfect fit for IoT applications and network systems. It can be programmed in Arduino or other IDEs and over its serial interface. An overlook of the board can be seen in Figure 2.

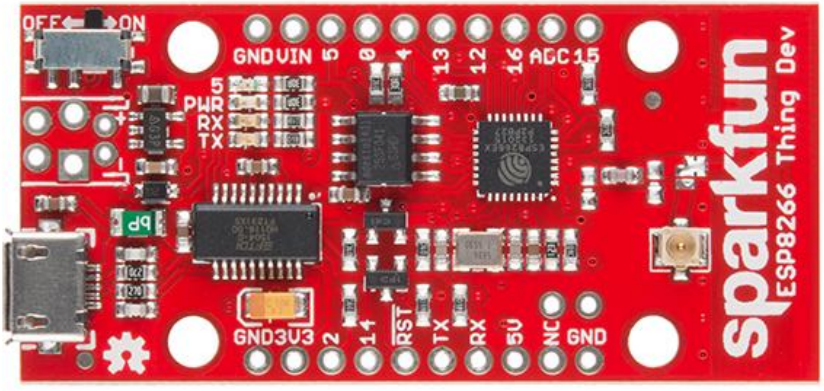

Figure 2 SparkFun ESP8266 Dev Thing [2]

The board has two parallel breadboard compatible pin rows. The upper part consists of the micro-USB connector for serial communication, a LiPo adapter for optional power supply, and ON/OFF switch and four LEDs indicating the power status, TX and RX signals status, and one connected to pin 5 for user control.

On the middle section, you will find a 32-bit RISC CPU: Tensilica Xtensa L106 running at 80 MHz (it can be overclocked to 160 MHZ), 64 KB of instruction RAM, 96 KB of data RAM and an External QSPI flash of 512 KB.

On the lower part of the board, a Wi-Fi PCB trace antenna is included, however and external U.FL-to-RP-SMA adapter/2.4GHz Duck Antenna can be connected to increase the range of the board. On the back side of the board, a RST jumper and a SLEEP-MODE connector can be found. This mode is used for ultra -ow power mode, when the board does not perform any processing tasks.

The ESP8266 operates with a voltage range of 3.7V to 6V. In USB mode, it takes in 5V. It can output, however, 3.3V and if USB is connected the PIN “5V” can output up to 4.8V. The PIN out table can be found in the Appendix 1 ESP8266 PIN connections

The ESP8266 Thing Dev is Wi-Fi 802.11 b/g/n capable. It has Wi-Fi Direct P2P communication abilities which enables devices to connect to each other without requiring a wireless access point. It supports multi-hop radio communications (See 2.2 Network Topologies), and has an integrated software Access Point (AP). The AP enables the device to be made a “router” via software. This is used as a gateway for the device network, and can be uses to access the internet.

The biggest advantages of the ESP8266, are its compact size, it measures 2.8 cm by 5.5 cm, and its Wi-Fi connectivity. However, given that certain application requires an extra layer of reliability and sometime better performance, the ESP8266 might fall short when it comes to ultra-speed processing.

2.3.2 HC-SR 04

Ranging sensors measure the distance to a target, using a variety of approaches. This work will focus on the ultrasonic solution. These sensors have high sensitivity, high accuracy and low hysteresis.

The ultrasonic ranging module [4] has a 2 - 400 cm non-contact range measurement ability. The accuracy resolution is 3mm. It has 3 main parts: the transmitter, the receiver, and the processing board. Figure 3 presents the module appearance.

Figure 3 HC SR-04 Ultrasonic ranging sensor module [2]

It measures 4.4 cm by 2 cm by 1.1 cm. There are only four pins on the module: VCC (Power), Trig (Trigger), Echo (Receive), and GND (Ground). the module operates at 5V DC. It works by triggering for at least 10µs HIGH level signal to the TRIG PIN. The module in turn, sends eight 40 kHz and detect whether there signal back or not. When a signal is detected, a HIGH level state is outputted to the ECHO PIN that last the time from sending to returning of the initial pulse. Figure 4 shows the operation of the module.

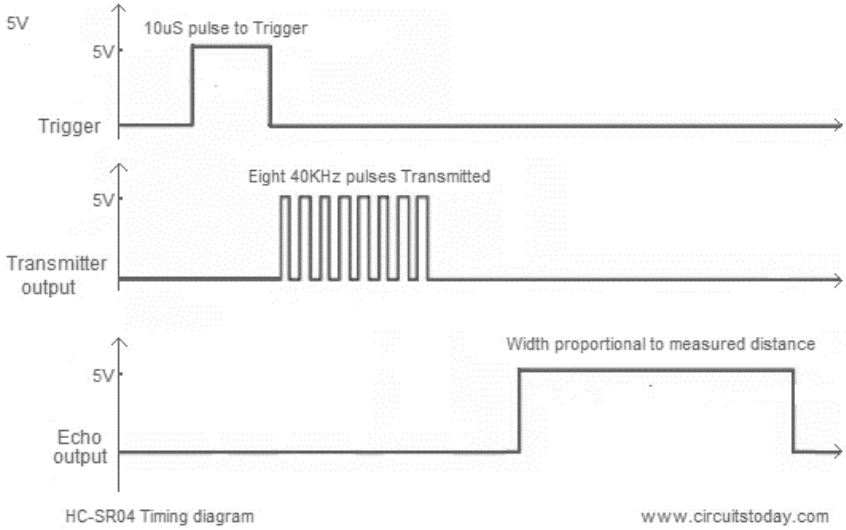

Figure 4 HC SR-04 Operation [4]

This module can be used for a wide range of applications. They can vary from tracking the position of individual objects, to pipe flow measurement, and position services. However, for specialized application that requires measurement speeds faster than 40Mhz, this module might not be suitable.

2.4 Software

Sometimes the sensor network could be deployed in an environment where energy or available space are scarce, hence an optimal software solution is required to maximize the power consumption, reduce de risk of failure, and increase the reliability and performance of the system.

2.4.1 Arduino

There are many software applications when it comes to sensor development. Nevertheless, in modern times, open source software, easy access information and open access communities, make Arduino one of the best options when it comes to development a fast and cheap prototype for visualization and testing of electronic systems.

Arduino is an open-source electronics platform based on easy-to-use hardware and software. [5] Arduino capable boards can perform input-output operations (I/O), by coding instructions into the MC.

Arduino is a relative inexpensive cross-platform solution. It has an integrated development environment (IDE) which uses a set of C/C++ language instructions to control the boards. It is open source and easily expandable with extra libraries and modules. [6]

2.4.2 UDP Protocol

The User Datagram Protocol (UDP) is one of the main components of the internet communication protocols. Originally designed in 1980 [7], this protocol can send or broadcast messages, called datagrams, to another host on an Internet Protocol (IP) network. It is part of the transport layer of the Open Systems Interconnection (OSI) model for communication protocols.

It differentiates from the Transmission Control Protocol (TCP), in the OSI model, because it does not require authentication or handshake from the sender to the receiver, hence the broadcasting capabilities, still, this also means that a duplicate protection and delivery reliability can be compromised.

The UDP header picture on Figure 5 demonstrates that the header consists of four fields, each of which is two bytes (16 bits). The source port identifies the sender's port, if the sender acts as client. The destination port denotes the "server" or receiver's target port. The light defines the size in bytes of the UDP data. The field size sets a theoretical limit of 65,535 bytes per datagram. The 16-bit checksum field is used for error-checking of the header and data [8].

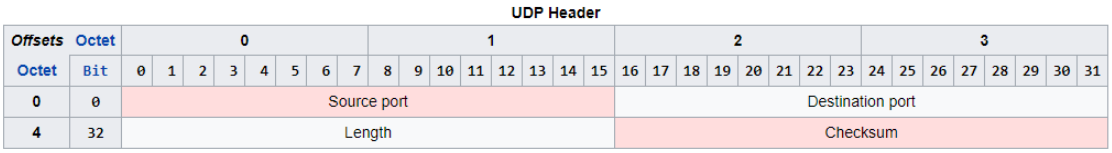

Figure 5 UDP header [9]

The main advantage of UDP is that, assumed it does have acknowledgement of transmission nor ordinance, and because it is built on top of the IP, it makes it incredible fast. This makes it quite useful for several internet applications, such as the Domain Name System (DNS), and the Dynamic Host Configuration Protocol (DHCP) [10]. Similarly, some virtual private networks (VPN) use UDP if error checking implementations at the application level.

2.4.3 Node.js

As an asynchronous event driven JavaScript runtime, Node is designed to build scalable cross-platform network server-side applications. Node.js runs server scripts to produce the website content before is sent to the renderer. This allows Node.js to be used as a web application development platform and unify system around a single programming language. A code preview can be seen in the Appendix 2 Node.js Sample Server.

Node.js works in this same asynchronous way. It uses a function called the Event Loop that ensures that data is not constantly being queried for, but simply transmitted when it

exists. [11] It operates on a single thread, which allows it to support thousands of users without requiring context switching i.e. multitasking processing.

It presents an Event Loop as a runtime construct instead of as a library. Node does not have a start of the loop event, which means that the loop is hidden from the user and its always active, this in turn creates a low latency system.

Node lacks vertical scaling, i.e. adding resources to a single node in a system, due to the single thread architecture. Nonetheless it cannot take advantage by forking or dividing the system as in multithreading, and allow these different entities to communicate with each other.

2.4.4 Vue.js

Vue is a progressive JavaScript framework for building user interfaces on top of a JavaScript runtime. Vue is focused view and rendering of the application, while also serves as a medium for user input, which allows it to be progressively adoptable. It can be used in existing projects and replace other frameworks completely [12].

Vue works by routing declarative data to the Document Object Model (DOM). All Vue values are HTML valid and can be rendered by any browser. Vue consist of three main elements: the components, the methods, and the properties. These three categories are dependent of a single Vue instance that contains all the data for a single page.

The component declares the different object of the page. Properties assign different values to these components, while the methods assign different behaviour to these objects.

The best quality of Vue is that its cross referenced. It is possible to have multiple objects with the same properties and change only one single value for all them. All these values and behaviour are plain JavaScript object, declared inside the HTML code, allowing each component to keep track of the source. The system has re-render capabilities, i.e. the system knows when and what have changed and the proceed to re-render these changes. A code preview can be seen in the Appendix 3 Vue.js Sample syntax.

.

Vue have several integrated features such as CSS transitions and animations, which can be declared in the same way as the JavaScript mark-up inside the HTML code, and a small 71 kB size for the whole script, which makes it easily portable for IoT applications that requires any user interface.

2.4.5 MongoDB

MongoDB is an open source database that uses a document-oriented data model. It is a non-relation (NoSQL) database which provides storage and retrieval of data different than the traditional Structured Query Language (SQL). This solution uses JavaScript Object Notation (JSON) which is an open standard file that is easy for humans and machines a like to read and parse. JSON consist of attribute-value pairs organized in an array data form [13].

Mongo uses JSON values and structures them in a way that can be changed over time. It supports indexing, replication, i.e. multiple instances of the same data, and aggregation, which can join multiple databases without having to index all of them all over again.

Mongo its distributed and supports horizontal scaling, i.e. more nodes are added to the network, and object mapping. However, when the databases exceed the 32-bit size the indexing can miss certain documents.

2.5 3D Printing Solutions

With the advent of faster processing systems and cheaper material, 3D manufacturing because a day to day uses in many applications. From industrial prototyping and automotive solution, to home basis objects that anyone can uses.

There are many technologies for 3D print an object. This work will focus in additive manufacturing(AM). This is a technique where layer of a specific material is built on top of each other to create a three-dimensional object.

The process starts by having a 3D reproduction of and object as a computer aided design (CAD) file. This file is a mathematical model of the object and have x, y, and z coordinates that the printer can understand.

Once the CAD model is ready, a slicer program is required. This software slices the segments the model and create 2-dimensional surfaces that once put together will render a 3D object in the real world.

There are many variables to consider before printing an object. The resolution of the print, the material, and printing technology can affect the look, the strength and the performance of an object. This work will focus on the different technologies, especially in fused deposition modelling (FDM) and its different approaches

2.5.1 FDM

Fused deposition modelling works by extruding a filament of thermoplastic material through a heated nozzle in a movable head. the material heats up and melts. Figure 6 depicts the head of a 3D printer. The head moves continuously along a predefined path in the x and y direction first. once a layer is completed the head moves up 1-layer width and starts the process all over again. This process builds the piece step by step, layer by layer. Depending of the complexity of the object support structures might be added, that will reinforce certain gaps in the model.

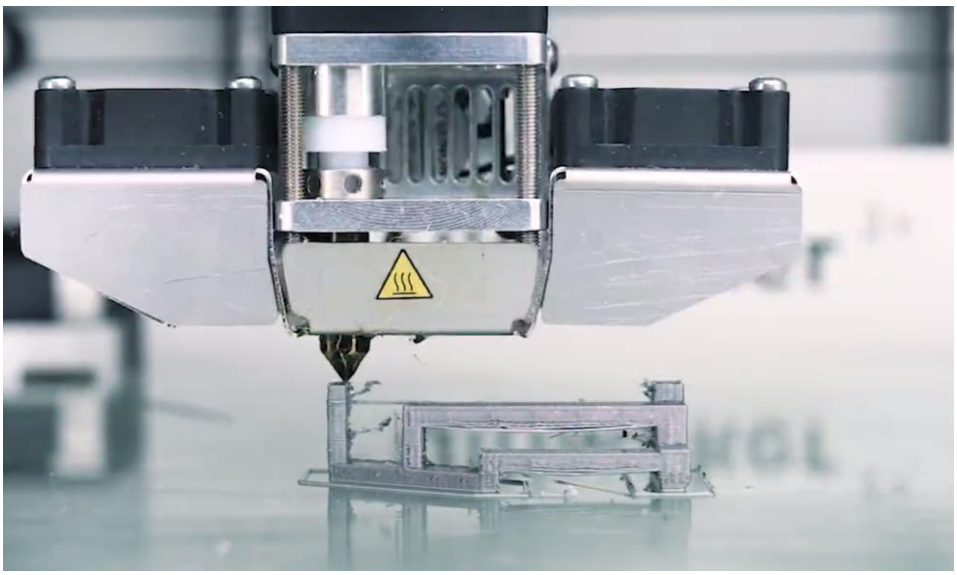

Figure 6 3D Printer head [14]

FDM process usually print the layer undersized, given that the material will compressed and adhere to the layer immediately below as show in the Figure 7. This process is controlled by the slicer software.

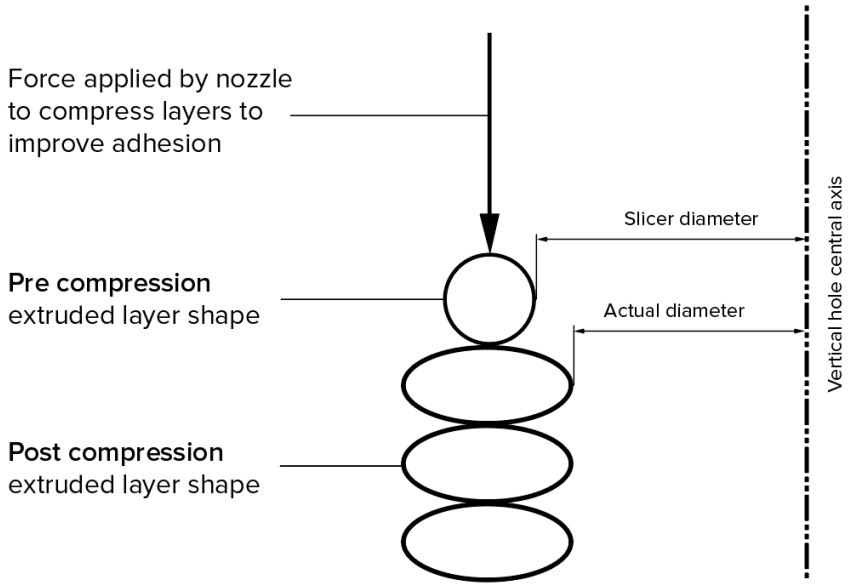

Figure 7 Layer deposition deformation [14]

There should be certain design consideration while using this process. Having overhangs can cause the material to deposit and fall in one side of the pieces. Also having bridges without support can cause a collapse in the structure. The corners must be post processed, because the material deposits and smooths as it cools down. Its recommended to build the piece in a parallel to the tension force of the application. This will give rigidity and stability to the piece. Regardless of the drawbacks FDM stays as the most popular type of 3D AM.

After printing, post processing can be used. Common materials include Acrylonitrile butadiene styrene (ABS), which have significant mechanical properties, such as impact resistance and toughness; polylactic acid or polylactide (PLA), which is easier to handle, its biodegradable and has better performance at sharp angles; and nylon, which is more versatile and flexible than PLA or ABS.

2.6 Modularity

A system can be scalable by combining smaller systems into a larger one. This is the main idea of modularity. In Sensor systems, smaller pieces join to a main “node”, which connects to a main server that gather the information. This node architecture is self-contained and can be reproduced any required amount of times.

When the system has modules containing these nodes, and all modules are the same, each with a specific identificatory, it becomes modular. Adding or subtracting modules will not affect the performance of the system, only its reach, if there is always at least one module.

The modules can be interchangeable and the system can identify and process these changes without losing functionalities. Modularity grants freedom, in a way that it could be used to build multi- application systems that all work can be configured as required, in controlled but fast manner.

3 Sensor Environment Design

One of the key features required for this work, is that the device should be as small as possible, considering that it must be discrete and of easy installation. It must have low power consumption and easy maintenance. For this purpose, several MC boards were pre-selected. At the end, the ESP8266 was adopted. Its small size, easy to power options and low cost, make it the best choice for this project.

Similarly, an easy way to manufacture and assembly was necessary. In this case 3D additive technology solution is necessary to create the support and casing of the prototype.

The server should listen and parse all the incoming data from all the different devices, and at the same time be able to run applications that will use this data. A strong framework that can handle these requirements while being lightweight, was chosen for this task; in this case Node.js and Vue.js

3.1 Device Core

The main core of the device consists of three parts. The ESP8266 board, the HC-SR 04 ultrasonic sensor and Reset (RST) button. The system is designed in such a way that no glue or extra soldering of the pieces was required. This allow for cost effective pieces replacement and easy development of the device.

3.1.1 Board

The connections to the board were made by soldering a 2.54 mmm right-angle pin connector to the board. The pins used are depicted on Table 2.

Table 2 Device selected PINs

ID	PIN Number	Description
TRIGGER	12	Sends HIGH level to the sensor
ECHO	13	Read HIGH level from the sensor
RST_BTN	4	Physical reset of the board
LED	5	Physical indication of operation
GND	GND	All modules share the same ground

The board have soldered connector, but no soldered connections. This was intentionally made this way to have a fast development and implementation.

3.1.2 Circuit

The circuit has few connections since all the operation happens inside the sensor itself and the processing in the board. However, an external reset (RST) button was required. Figure 8 shows the final circuit and the components values and connections.

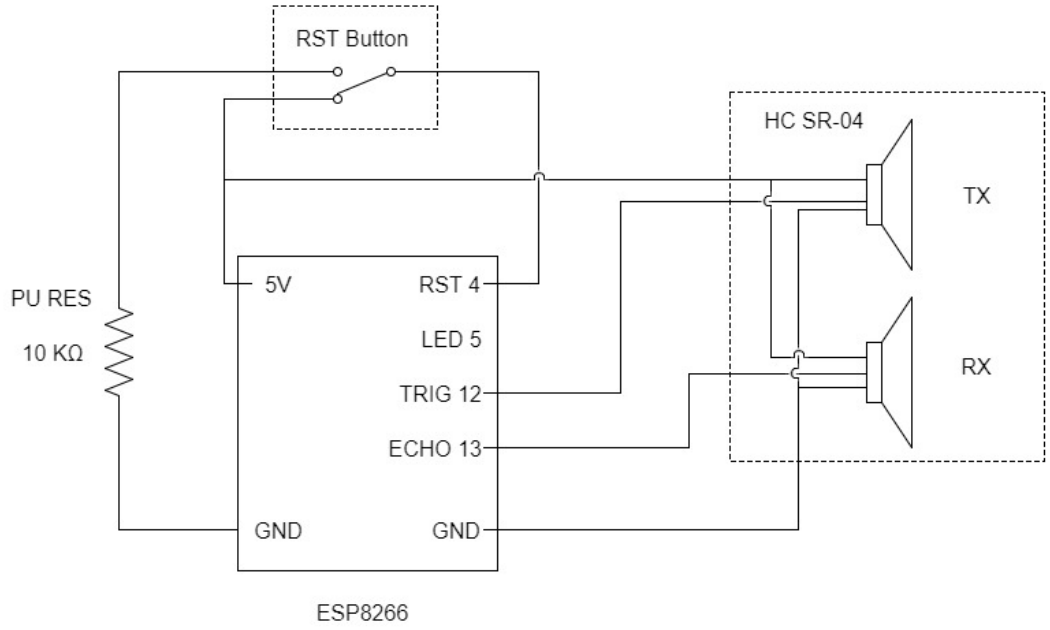

Figure 8 Circuit diagram

A pull up resistor was required to connect the RST button setting a LOW level to PIN 4 hence triggering the reset function of the board. The system is powered over USB to wall adapter, giving a constant 5V DC to the system

3.2 Program

The basic operation of the ESP8266 requires two different yet complementary parts. First the client, which is coded into the board; and second, the server, which registers, parses and process the data from the devices. The ESP8266 can be programmed with the basic Arduino IDE, though VS code was used in this case. In both cases the implementations are the same.

3.2.1 Client

The client consists of two main processes that are running constantly: the sensor data acquisition and the Wi-Fi client connection check-up. Similarly, there are two main functions that are executed on the start of the device, i.e. the EEPROM system for Wi-Fi credentials storage, and the Access Point (AP) handler, that setups the connection to the main Wi-Fi network, and creates its own network when configuration data is required.

Once the device is powered on, the EEPROM is checked to see if there is any Wi-Fi network registered. The system checks for each position of the EEPROM, 32 for the SSID name, 32 for the password, and finally for the IP (4 bytes) of the server within the network. After the EEPROM has been scanned, the next function is executed. Figure 9 demonstrates how this process works.

```
101   if ( ee_ssid.length() > 2 ) {                  // Starts WIFI connection
102     WiFi.begin(ee_ssid.c_str(), ee_pass.c_str());
103     Serial.println();
104     Serial.println("Waiting for Wifi to connect...");
105     if (!connectDEVICE()){
106       setupAP();                                   // Calls AP if no connection
107     }
108   }
109   else {
110     setupAP();                                     // Calls AP if EEPROM empty
```

Figure 9 Wi-Fi Connection Handler

The board invokes the `Wifi.begin()` command which tries to connect to the specified SSID. Once the connection is established, the main loop will be executed. After an automatic timeout of 10s, if the connection could not be created the function `setupAP()` is called. This function disconnects and turns off all the board's Wi-Fi connections, if any, and proceed to turn the ESP8266 into workstation mode, which in turn will scan the area for active connections and will display the SSID, the encryption type and the relative Received signal strength indication (RSSI), and the quality of the signal for each found network, as well as creating its own AP.

The AP consists of a broadcasting service created by the ESP8266 which allows input via HTML rendering. Once created, from any device with Wi-Fi enable access e.g. smartphone, laptop, etc, it is possible to configure and input the require data for the ESP8266. Figure 10 demonstrates how the AP looks on a PC.

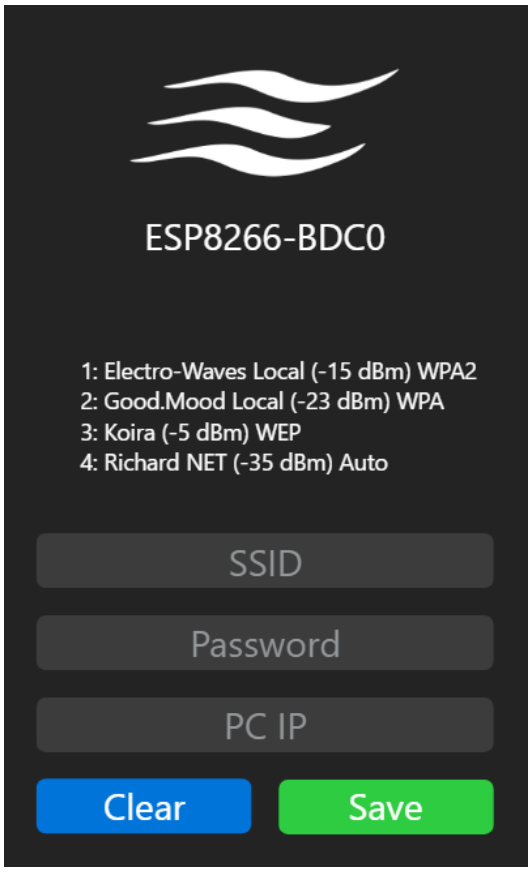

Figure 10 Client render

The user interface is done with HTML and CSS. Given that the usable memory of the ESP8266 is less than 200 kB, all the elements on the web client are rendered in-browser. The actual code for the HTML occupies less than 2 kB and the CSS less than 1 kB. The EW logo is rendered as an SVG, which is a math-based XML-based vector image format, pulling the size of the image from around 50 kB to less than 3 kB.

Below the logo, the name of the device in this case the board model and last four digits of the MAC address are displayed. Under this, the results of the scanning made by the `setupAP()` function are shown. The results are displayed solely to allow the user to check if their target network its being detected by the board or not.

The user must input the desired SSID and its password, and the target server IP. The user can choose between “CLEAR” the EEPROM of the device (in case that another undesired network its stored in the device), and “SAVE” the entered information. This will save the data to the EEPROM, and restart the device, which in turn will check if the configuration its valid and try to connect to the required network. Appendix 4 Client Flow chart depicts the client functions and processes.

The system has a function that checks if its connected to the stored network. It can be set up to execute this function, from 1 to 1800 seconds, given the timeout of the board.

3.2.1.1 Sensor Data

Immediately after a stable connection has been achieved, the system will start to send the data acquired from the sensor. The process `void loop()` consist of two parts. The first one checks if the RST button is pressed. This button sends a HIGH state to the pin 4 of the board that tells the program to execute the `clearANDrestart()` function, if the button its pressed for more than 5 seconds. This function will erase all the data from the EEPROM and reset the device, prompting it into AP mode.

The second and most relevant part of the whole system is the acquisition and parsing of the sensor data. First the ESP8266 sends a digital HIGH to the pin connected to the HR-SC 04, in this case `TRIGGER PIN 12`, for 10 ms. Then goes to a LOW state. This causes the HR- SC 04 to emit its ultrasonic burst. The reflected sonic wave will be detected by the sensor (if it is within range) and an echo signal will be sent to the ESP82666.

The distance is given by Equation 1, where *d* is the distance, *t* is the duration of the pulse at the input pin, in this case `ECHO PIN 13`, *ss* is the speed of sound at sea level at 20 °C, which is equal to 340.29 m / s. It is divided by two because the wave travels twice the distance of the target measurement.

$$d = \frac{(t \times ss)}{2} \quad (1)$$

Equation 1 Distance calculation

The distance value is then converted into ASCII format and assigned to the `char` variable. When the device successfully connects to a network, it saves its MAC address into a `char` variable as well. Each device has an exclusive MAC address, and each device can be identified with it. Figure 11 Data Parsing and UDP transmission commanddisplays the parsing of the of the data. The function `sprint()` assembles the data from the distance and the MAC address into a single string in JSON format.

```
398     ltoa(distance, DATA_DIST, 10);                  // Converts long to ascii
399
400     sprintf(DATA_TX, "{\"mac\":\"%s\",\"dist\":%s}", DATA_MAC, DATA_DIST); // Sensor Data
401
402     Serial.println(DATA_TX);
403
404     UDP.beginPacket(SERVER_IP, localPort);          // Starts UDP Send
405     UDP.write(DATA_TX);
406     UDP.endPacket();                                // Ends UDP Send
```

Figure 11 Data Parsing and UDP transmission command

Finally, the `UPD.begin` command proceeds to send the package via the UDP protocol over its Wi-Fi connection to the previously entered target server. The local port can be changed, but its predefined to be 8888.

The `void loop()` has a delay of 30 seconds, i.e. it sends the value acquired from the sensor every 30 seconds to the server. This delay can be changed according to the requirements if it falls within the timeout time of the board.

When the board has lost its connection, the system will stop broadcasting the data and it will enter in re-connect mode. Here the system will try to re-establish the connection with the saved network for a given amount of time, after that it will go into AP mode and ask for new credentials.

3.2.2 Server

The server uses vue.js framework for rendering, node.js for the server and processing, and MongoDB for handling the device database. It has two main purposes, first as a hub for all the devices where the processing is made, and second, as a platform where different applications can be built on top on.

The processing of the data begins as the server gets the different JSON strings via UDP from all the devices. Each transmission has a different MAC address, this will allow the server to assign a different role and ID to each device within the network. Figure 12 depicts the parsing of the incoming JSON and their assignment.

```
// UDP Server on message
server.on('message', (msg, rinfo) => {
  console.log(`server got: ${msg} from ${rinfo.address}:${rinfo.port}`);

  // Convert message to object
  let data = JSON.parse(msg);

  // If device isn't in the devices array: Add it to the database and add it to the array
  if (!checkDevices(data.mac)) {

    // Save to the database
    db.collection('devices').save({ mac: data.mac }, (err, result) => {
      if (err) {
        console.log(err);
      }
      else {
        // Push the data to the devices array
        devices.push({ mac: data.mac, configured: false });
        console.log(data.mac + 'saved to database')
      }
    })
  }
```

Figure 12 Server parsing

Fist the server gets the message from the specified IP and port (which must match the ones settled at the client side). Afterwards it checks if the MAC address is present in the database, if not, it proceeds to list it without configuration. If the device is indeed present, it checks it configuration and current assigned state. The server has a build-in password-backed administrator page where all the active devices are listed. Figure 13 illustrates the main user interface of the server.

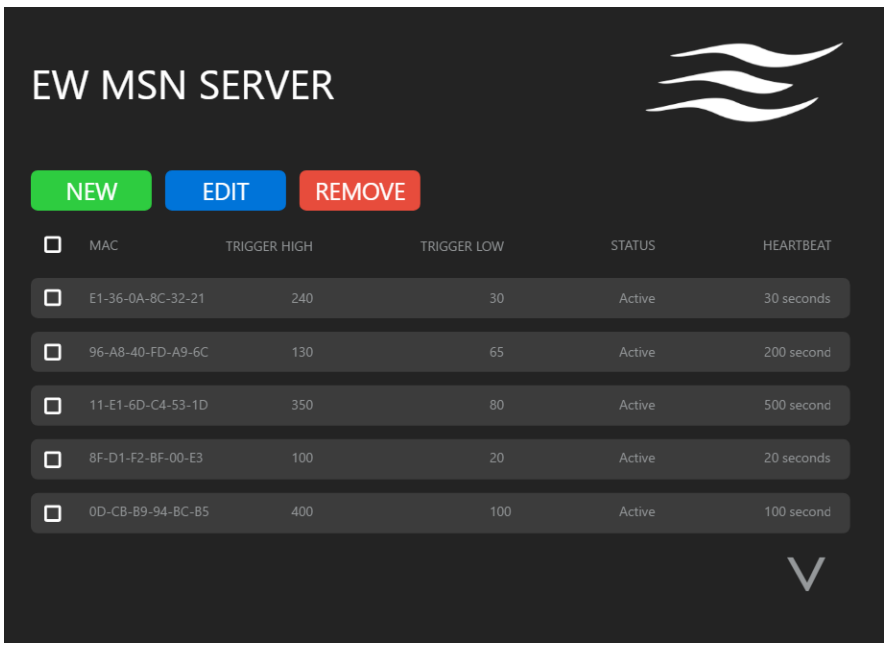

Figure 13 Server main UI

New devices can be added by clicking the “NEW” button. It prompts a list of un assigned active devices and where the user might add them to the system. The “EDIT” button prompts the edit mode. Here the User can change the configuration of the devices, such as the trigger range, and the assignment. When a device is no longer required or has been deactivated, the “REMOVE” button deletes it from the system, with all its current configuration.

The heartbeat indicates the last known time when data was received from a device. The current value shows the current reading in centimetres from a device. Finally, the Status shows if the device current state of a device.

The server framework lists and manages the devices, but it does not have any predetermined applications. Depending on the requirements vue.js and other JavaScript applications can be built on top of this framework. The applications will simply parse the data in the server’s database and use them as the customer requires.

3.3 3D Prototype

Having the programming of the client and the server done, a casing for the board and sensor was required. The design had to be as small and compact as possible without sacrificing easy assembly capabilities, nor interfering with the operation of the sensor. The casing consists of two parts. The bottom holds the board, the sensor, and the circuit., while the top part allocates space for the different connections and that upper part of the sensor. Figure 14 depicts the first sketches of the casing.

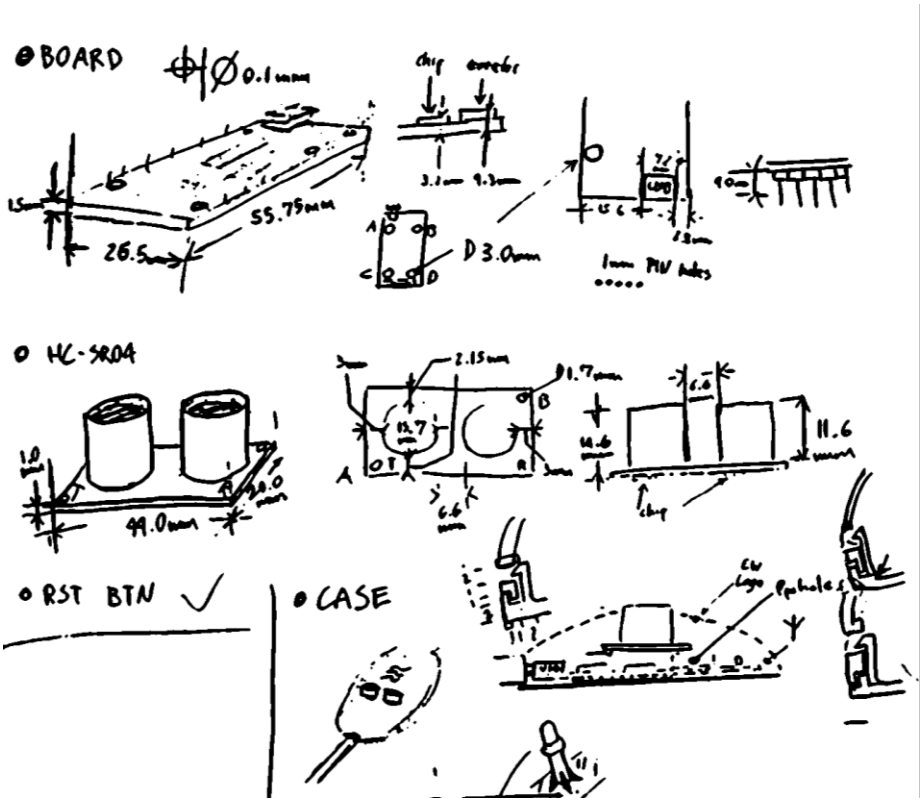

Figure 14 Casing sketches

The modelling of the bottom part had to include supports for the in-board holes of the ESP8266 and the HR SC 04 sensor. A snap support was made for the RST button circuit, as well as snap-fits for the top part. Figure 15 depicts the final model of both parts of the device

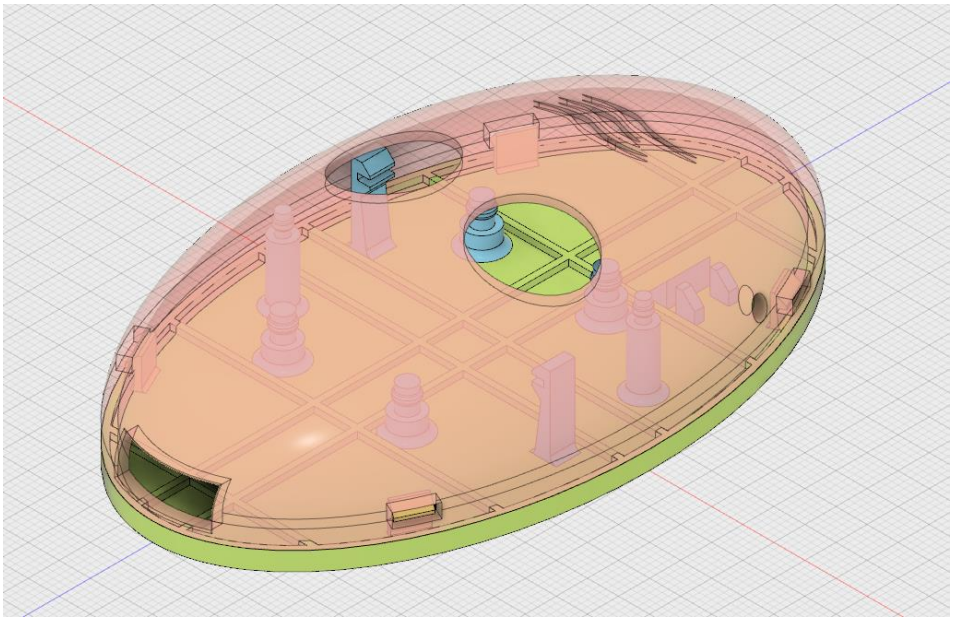

Figure 15 Final 3D model

The selected manufacturing process was 3D FDM, using PLA as material for the both parts. The bottom part was reinforced with a central 2-piece spine and several ribs to give extra rigidity given that this part must hold all the pieces together. In addition to this, the bottom part has an inner lip that helps with alignment between the two parts [14]. Furthermore, the bottom part includes four snap-fits, designed in way that does not interfere with the ESP8266, and allows an easy assembly of the system.

The top part has the two holes for the sensor on the upper part. USB connection space at the bottom, an RST button hole on the side, and matching snap-fits along the lateral edge. It also has an outer lip that coincides with the bottom part's inner lip, reducing the chance of gaps between the parts.

Both parts were designed with the least amount of straight edges as possible. The AM FDM PLA solution works better for smooth edges, given that the material tends to deposit with time, which gives smooth or rounded corners. Also, this gives circular rigidity to the system and it is easier to post process the parts. Figure 16 presents the final prototype of the device. Figure 17 shows the prototype without the white cover. Figure 18 is an example of how the device can be installed.

Figure 16 Final printed prototype

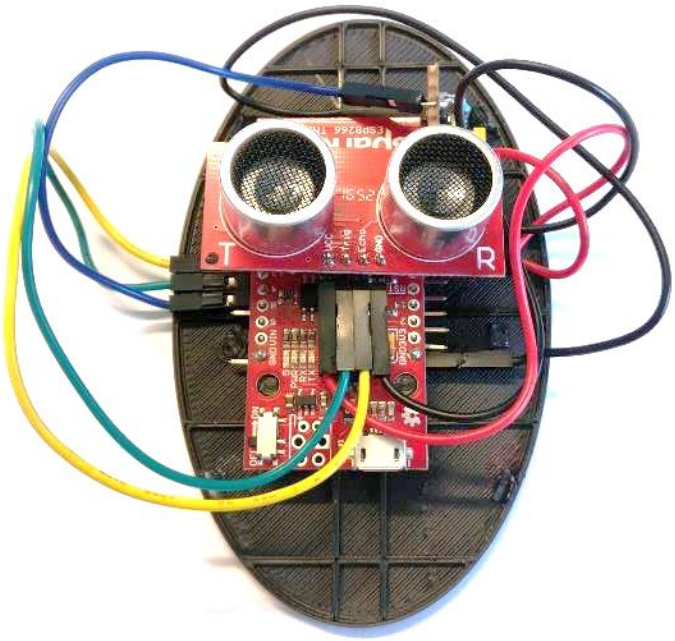

Figure 17 Cover free prototype

Figure 18 Wall mounted prototype

It measures 6 cm by 10 cm by 5 cm, and weights around 50 grams. It can be installed with through-hole screws or with additive materials. The device is designed to look user

friendly and non-invasive. The material chosen was PLA which gives it rigidity and is easy to replace and manufacture.

3.4 Sensor Environment

With the device's hardware and software completed, the next step was to assemble and test the environment. Figure 19 shows the environment final arrangement.

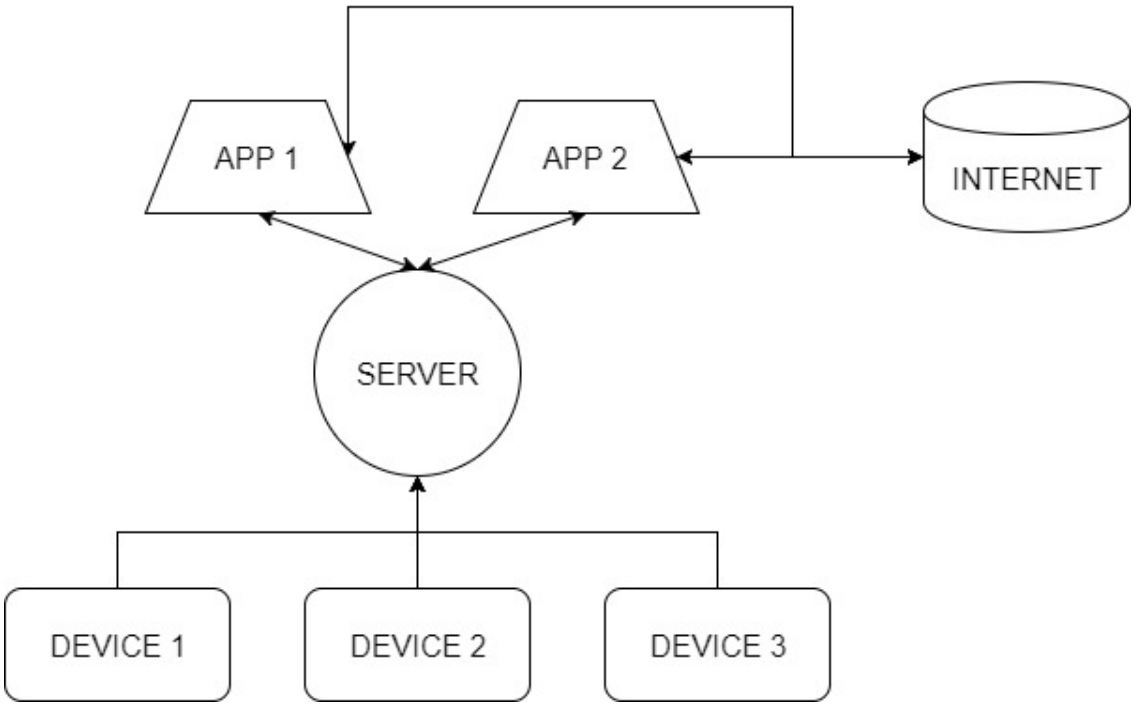

Figure 19 Environment structure

The environment is made of: the device(s) as client, the server running on a PC, and the application running on top of the server. It is easily configurable and modular, i.e. different applications can be mounted on the server and switch between as requested.

3.5 EW Reception

Only one device was fully manufactured, however it was possible to simulated a significant number of devices with software the following are the EW specific application and other AV implementations.

Electro Waves was lacking a centralized reception area for its customers, thus the task was to implement a system that will welcome the visitor, and help them navigate the premises, and deal with them while an employee comes for them.

The system is mounted on a PC or player and runs an application on top of the MSE. It consists of a stand and an interactive screen which displays the office premises with the status of the personal. Figure 20 presents the installed system.

Figure 20 Display stand running the EW reception

There is a device in each relevant office, and each of them has an assigned employee. This is displayed on the main screen on top of the office map. The system also displays information about EW, the current weather, and a RSS news feed. When a visitor gets closer to the stand, the screen turns on automatically and starts to display the information. The visitor can choose to send a notification of arrival to the required employee, which will in turn receive an alert that he/she is required at the entrance of the premises.

Figure 21 Showcases the main UI of the system. (For privacy and intellectual property reasons some features have been changed or are not shown in the following figure.)

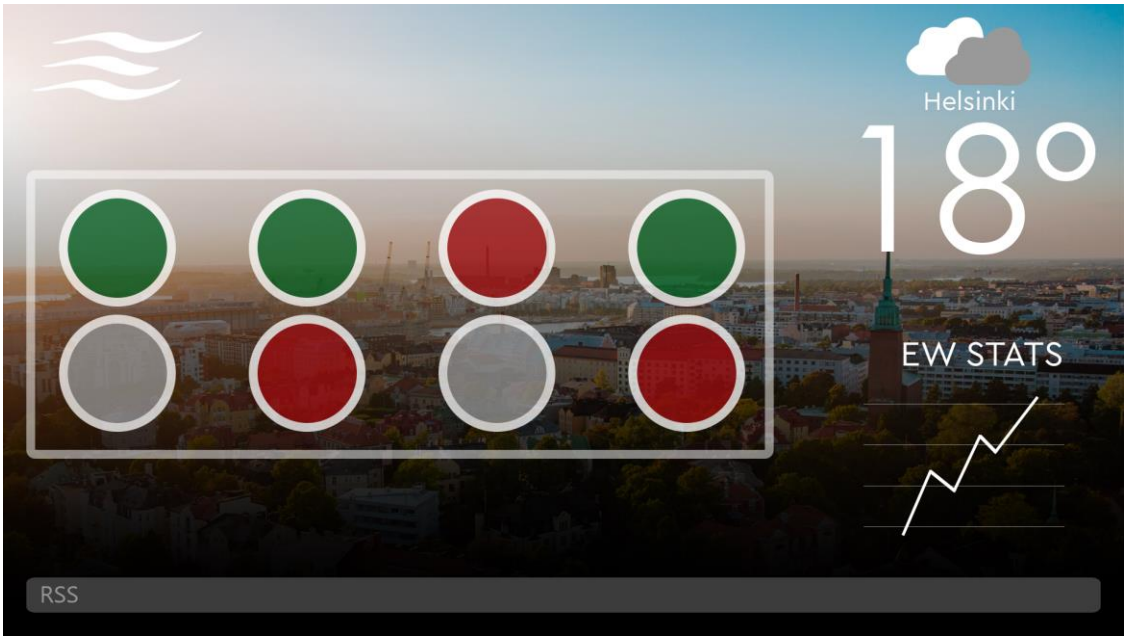

Figure 21 EW Reception main UI

If the visitor steps out of the range of the stand's sensor, the screen will turn off and go into idle mode.

4 Discussion and Results

The overall development of the system was reasonably straight-forward. The bottom-up design approach gives a clear view of the structure of the whole system. However, there are some matters that must be considered.

First, the systems could have been developed more robustly in a sense that for now it works only with one type of sensor at a specific voltage and type of signal. Second the fact that the board is not soldered to the other components give it modularity and easy maintenance, but it would have been preferable that the whole circuit is integrated into one single component. Further research in this aspect will be done for another iteration of the MSE. As mentioned before, the system uses a star-based network topology and the devices cannot communicate between each other yet. A future development will take account of this and implement a mesh-type topology, which will allow the system to use location triangulation techniques.

Programming took the most part of the time, given that several frameworks and technologies were used for the different parts of the system, but the communication between them was competent. Further optimization and iteration would require developing an integrated platform which can handle a greater number of devices. This will require the devices to be tethered to the power source. The ESP8266 has an integrated LiPo battery port, which might be used to deal with this limitation.

Currently the system operates on a wireless local area network (WLAN) and it is not connected to the internet. In future development, this issue will be addressed.

4.1 Privacy

There is an issue that requires special attention. It deals with the privacy concerns of the users regarding the reach of the system.

The proposed MSE is built in such a way that it can be extended over a vast area. It could be able to track moving objects by triangulating the different data from the sensors. The users or in this case objects, would be tracked and monitored all the time. Which leads to the question, how big can the system get, without invading the privacy of the customers? A study made by TSHEETS [15] concluded that around 70% of the employees accept that the employer knows their location at any time. However, it might be noticed that at the end it is up to the agreement that the company has with the employees and how the data is handled.

5 Conclusions

Using different framework and manufacturing techniques an MSE system was developed. The proposed system uses an ESP8266 MC and a HR SC-04 ultrasonic module for object detection and ranging. The device is powered via USB via a ACDC wall adapter and it communicates to the server via Wi-Fi using UDP protocol- based JSON messages. The device enclosure is made using AM FDM 3D printing technology for easy and reliable manufacturing.

The server is developing with Node.js, Vue.js, and MongoDB. It listens to the messages send by the device, parses the information, and stores the device's ID on a JSON-based database. Applications can be built on top of the server, that uses the store information for data visualization.

Beyond the limitations, the results were satisfactory. This system can be used for IoT and automation systems. Further projects of EW will have automatic interaction with EW video screens and embedded sensor for lightning and water systems products included. The proposed system has an immense potential for applications and it will be used as a template for future automated development.

References

[1] Mamatva, “Network topology ~ LMN technohub,” 2015. [Online]. Available: https://lmntechnohub.blogspot.fi/2015/01/network-topology.html. [Accessed 2017].

[2] SparkFun Electronics, “SparkFun ESP8266 Thing - Dev Board,” 2016. [Online]. Available: https://www.sparkfun.com/products/13711. [Accessed 2017].

[3] Espressif Systems, 2017. [Online]. Available: espressif.com. [Accessed 2017].

[4] SparkFun Electronics, “SparkFun HC SR 04 Datasheet,” 2015. [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf. [Accessed 2017].

[5] Arduino Inc, “Arduino - Introduction,” 2015. [Online]. Available: https://www.arduino.cc/en/Guide/Introduction. [Accessed 2017].

[6] Arduino Inc, 2015. [Online]. Available: https://www.arduino.cc/en/Guide/Environment. [Accessed 2017].

[7] D. P. Reed, 1980. [Online]. Available: https://tools.ietf.org/html/rfc768. [Accessed 2017].

[8] IPv6.com, “UDP - User Datagram Protocol | IPv6.com,” 2006. [Online]. Available: https://www.ipv6.com/general/udp-user-datagram-protocol/. [Accessed 2017].

[9] WikiMedia Inc, “User Datagram Protocol,” 2015. [Online]. Available: https://en.wikipedia.org/wiki/User_Datagram_Protocol. [Accessed 2017].

[10] RTC Magazine, “Speed Communications for Selected Applications with UDP | RTC Magazine,” 2012. [Online]. Available: http://rtcmagazine.com/articles/view/102819. [Accessed 2017].

[11] L. ORSINI, “What You Need To Know About Node.js - ReadWrite,” 2013. [Online]. Available: http://readwrite.com/2013/11/07/what-you-need-to-know-about-nodejs/. [Accessed 2017].

[12] Vue Org, “Introduction — Vue.js,” 2017. [Online]. Available: https://vuejs.org/v2/guide/. [Accessed 2017].

[13] JSON ORG, 2014. [Online]. Available: http://www.json.org/. [Accessed 2017].

[14] 3D Hubs, “3D Hubs Knowledge Base,” 2016. [Online]. Available: https://www.3dhubs.com/knowledge-base/cad-modeling-1-enclosures. [Accessed 2017].

[15] T Sheets Inc, “What Do Workers Really Think About GPS Monitoring?,” 2016. [Online]. Available: https://www.tsheets.com/gps-survey. [Accessed 2017].

[16] 3D Hubs, “How to design parts for FDM 3D Printing | 3D Hubs,” 2016. [Online]. Available: https://www.3dhubs.com/knowledge-base/how-design-parts-fdm-3d-printing#vertical-axis-holes. [Accessed 2017].

Appendix 1 ESP8266 PIN connections

Pin Label	ESP8266 I/O Function(s)	Notes
GND		Ground (0V).
3V3		3.3V
2	GPIO2, SDA	Can either be used as ESP8266 GPIO2 or I^2C serial data (SDA).
14	GPIO14, SCL, SCLK	Can either be used as ESP8266 GPIO14 or I^2C serial clock (SCL). Also used as the SPI clock (SCLK).
RST		The ESP8266's **active-low** reset input. The board includes a 10kΩ pull-up resistor on this pin.
TX	GPIO7, TX1	ESP8266 UART1 data output.
RX	GPIO8, RX1	ESP8266 UART1 data input.
5V		USB supply output. If USB is connected, this pin will supply about 4.8V.
NC		Not connected to anything.
GND		Ground (0V).
VIN		USB connected: ~5V output Can alternatively be used as a voltage supply input to the 3.3V regulator.
5	GPIO5	This pin is also tied to the on-board LED.
0	GPIO0	
4	GPIO4	
13	GPIO13, MOSI	Hardware SPI MOSI
12	GPIO12, MISO	Hardware SPI MISO
16	GPIO16, XPD	Can be connected to reset to wake the ESP8266 from deep sleep mode.
ADC	A0	A 10-bit ADC with a **maximum voltage of 1V**.
15	GPIO15	

Appendix 2 Node.js Sample Server

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Appendix 3 Vue.js Sample syntax

HTML

```
<div id="app">
  {{ message }}
</div>
```

JS

```
var app = new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue!'
  }
})
```

Appendix 4 Client Flow chart